

New Results for Adaptive and Approximate Counting of Inversions

Saladi Rahul

Dept. of Computer Science and Engg., University of Minnesota Twin-Cities,
4-192 Keller Hall, 200 Union St. S.E., Minneapolis, MN 55455, USA
sala0198@umn.edu

ABSTRACT

Counting inversions is a classic and important problem in databases. The number of inversions, K^* , in a list $L = (L(1), L(2), \dots, L(n))$ is defined as the number of pairs $i < j$ with $L(i) > L(j)$. In this paper, new results for this problem are presented:

1. In the I/O-model, an *adaptive* algorithm is presented for calculating K^* . The algorithm performs $O(\frac{N}{B} + \frac{N}{B} \log_{M/B}(\frac{K^*}{NB}))$ I/Os. When $K^* = O(NM)$, then the algorithm takes only $O(\frac{N}{B})$ I/Os. This algorithm can be modified to match the state of the art for the comparison based model and the RAM model.
2. In the RAM model, a linear-time algorithm is presented to obtain a tight estimate of K^* ; specifically a value which lies with high probability in the range $[(1 - \frac{\log N}{N^{1/4}})K^*, (1 + \frac{\log N}{N^{1/4}})K^*]$. The state of the art linear-time algorithm works for the *special* case where L is a permutation, i.e., each $L(i)$ is a distinct integer in the range $[1, N]$. In this paper, we handle a *general* case where each $L(i)$ is a real number.

1. INTRODUCTION

In this paper we revisit the classic database problem of counting inversions. The number of inversions, K^* , in a list $L = (L(1), L(2), \dots, L(N))$ is defined as the number of pairs $i < j$ with $L(i) > L(j)$. Each value $L(i)$ is a real number.

1.1 Motivation

Classical motivation. Interest in studying the counting inversions problem has been shown by various communities in computer science. It is considered an important measure to test the “sortedness” of the data. For example, sorting data is a critical operation in large-scale applications. Typically, such applications have multiple sorting algorithms and they perform some “tests” on the data to decide the most suitable sorting algorithm (an insertion-sort type algorithm is fast if the data is almost sorted). One of the important test happens to be counting inversions. We refer the reader to the book of Knuth [15] and the survey report of Estivill-Castro and Wood [11] for a detailed discussion on how counting inversions is crucial to the engineering of a fast sorting algorithm.

Modern motivation. Modern applications have revised

the interest in the problem of counting inversions. We briefly mention the applications here: (a) The number of inversions between two permutations is important for *rank aggregation* in Internet-based applications [9], and (b) The robustness of a *ranking function* (of database entries) can be tested via counting inversions. We strongly refer the reader to Ajtai *et al.* [3] for a nice detailed description of how modern applications benefit from counting inversions.

1.2 Previous work on counting inversions

Non-adaptive algorithms. The standard textbook solution for the counting inversions problem takes $O(n \log n)$ time by mergesort. There have been improvements over the $O(n \log n)$ time algorithm in the RAM model. Using Dietz’s dynamic ranking structure [8] counting inversions can be done in $O(n \log n / \log \log n)$ time. Few years back, Chan and Patrascu [6] could significantly improve the running time to $O(n \sqrt{\log n})$. Interestingly, in the RAM model counting inversions seems to be harder than sorting: the best known deterministic sorting algorithm takes $O(N \log \log N)$ time [13] and the best known randomized sorting algorithm takes $O(N \sqrt{\log \log N})$ expected time [14].

Adaptive algorithms. One approach to develop faster algorithms is to build solutions which adapt based on the number of inversions. Mehlhorn [17] presented an $O\left(N + N \log\left(\frac{K^*}{N}\right)\right)$ time algorithm to count inversions in the comparison based model. Adapting the approach of Pagh, Pagh, and Thorup [18], Elmasry [10] presented an $O\left(N + N \sqrt{\log\left(\frac{K^*}{N}\right)}\right)$ time algorithm in the RAM model.

Approximate algorithms. The other approach to develop faster algorithms to count inversions is to approximate the value of K^* . To obtain faster algorithms, Andersson and Petersson [4], and Chan and Patrascu [6] studied the approximate version of counting inversions problem. If the number of inversion in the list is K^* , then their algorithm will report a value within an additive error of εK^* .

Streaming setting. The focus of this paper is the RAM model and the I/O-model. However, there are other interesting models in which this problem has been studied. For example, the last decade saw the streaming community getting interested [3, 12].

1.3 Our Results

In this paper, we present two new results on the problem of counting inversions.

Adaptive algorithm. In the I/O-model we present an adaptive algorithm which counts the number of inversions using $O(\frac{N}{B} + \frac{N}{B} \log_{M/B}(\frac{K^*}{NB}))$ I/Os. Previously, such adaptive algorithms were known only in the comparison model [17] and the RAM model [10]. Neither of these solution can be trivially modified to work efficiently in the I/O-model. For example, adapting the algorithm of [10] to the I/O-model requires $\Omega(N)$ I/Os, since it inserts one point at a time. Interestingly, our algorithm can be modified to match the state of the art for the comparison based model and the RAM model. In that sense, our algorithm subsumes the results of [10, 17]. Please see the appendix for a brief description of the I/O-model.

Approximate algorithm. This problem is studied in the RAM model. We present an $O(N)$ time algorithm which reports a value in the range $\left[\left(1 - \frac{\log N}{N^{1/4}}\right) K^*, \left(1 + \frac{\log N}{N^{1/4}}\right) K^*\right]$. The estimate is correct with probability $1 - 1/N^c$, where c is a constant independent of N .

Chan and Patrascu [6] also presented an $O(N)$ time algorithm for this problem. However, their solution works only for the special case where L is a permutation, i.e., each $L(i)$ is a distinct integer in the range $[1, N]$. Because they consider a permutation, they make use of the Spearman's Footrule [7] which already gives a 2-factor approximation of K^* . In this paper, we study the more challenging setting where each element in $L(i)$ is a real number. A new approach is needed to handle this setting.

2. RED-BLUE DOMINANCE COUNTING

We start by defining the *red/blue dominance counting problem*. We are given a red list $R = (R(1), R(2), \dots, R(n))$ and a blue list $B = (B(1), B(2), \dots, B(n))$. Each element in R is mapped to a two-dimensional point: $R(i)$ is mapped to a point $(i, R(i))$. Similarly, each element, say $B(i)$, in B mapped to a point $(i, B(i))$. A pair (r, b) is called an *domination pair* if r is a red point dominated by a blue point b . A blue point b dominates a red point r if b has a larger x -coordinate than r and b has a smaller y -coordinate than r (see Figure 1(a)).

Throughout the paper, we will interpret R and B as one of the following: (1) a list of N elements storing real-values, or (2) a pointset in two-dimensional plane. It will be clear from the context which interpretation is being taken.

Let K^* be the number of domination pairs in R and B . Counting inversions is a special case of this problem by letting the red point set be equal to the blue point set. In this paper, we present two results for the red-blue dominance counting problem.

THEOREM 1. (Adaptive algorithm) *Red-blue dominance counting problem can be solved using*

$O(\frac{N}{B} + \frac{N}{B} \log_{M/B}(\frac{K^*}{NB}))$ I/Os, where K^* is the number of domination pairs. When $K^* = O(NM)$, then the algorithm uses only $O(\frac{N}{B})$ I/Os. This problem is studied in the I/O-model.

THEOREM 2. (Approximate algorithm) *Red-blue approximate dominance counting problem can be solved in $O(N)$ time. For a fixed constant c , with probability $1 - 1/N^c$ the algorithm will report a value in the range $\left[\left(1 - \frac{\log N}{N^{1/4}}\right) K^*, \left(1 + \frac{\log N}{N^{1/4}}\right) K^*\right]$. This problem is studied in the RAM-model.*

In Section 3 we will define the concept of red-blue cells along with their properties. At first look, it might not be clear to the reader as to why we need red-blue cells. Then in Section 4 we will make use of them to obtain the adaptive algorithm and then in Section 5 we will use them along with random sampling techniques to obtain the approximate algorithm.

3. CONSTRUCTION OF RED-BLUE CELLS

Given the lists R and B , and a parameter K , we want to construct a set of *red-blue cells* C_1, C_2, \dots, C_ℓ . A red-cell is a rectangle of the form $(-\infty, x) \times (y, \infty)$, and a blue-cell of the form $(x, \infty) \times (-\infty, y)$. With each cell C_i we associate a set of red points $R_i \subseteq R$ and a set of blue points $B_i \subseteq B$. Consider the following two cases:

(1) If $K^* \leq K$, then we want to construct red-blue cells which satisfy the following *three* properties:

- A) $\forall i \in [1, \ell], \min\{|R_i|, |B_i|\} = O(K/N)$.
- B) For every domination pair (r, b) there will exist exactly a single integer i such that $b \in B_i$ and $r \in R_i$.
- C) $\sum_{i=1}^\ell |R_i| = O(N)$, and $\sum_{i=1}^\ell |B_i| = O(N)$.

(2) If $K^* > K$, then we either construct the cells with the properties described above, or we are allowed to report a *failure*.

LEMMA 1. *The red-blue cells can be constructed in $O(N/B)$ I/Os.*

The rest of this section is dedicated to proving Lemma 1.

3.1 First step: Red cells

Shallow cuttings for various geometric objects are widely used in computational geometry to answer range searching and related problems (for example, [1, 16]). Shallow cuttings as described in this section have been used before by Vengroff and Vitter [19]. On the technical side, our key contribution is a novel and a non-trivial application of shallow cuttings.

Consider a red pointset R . Informally, a k -shallow cutting on the pointset R has the form of a “staircase” which is a one-dimensional, monotone sequence of orthogonal line-segments. Formally, a k -shallow cutting is a curve $C = c_1 d_1 c_2 d_2 \dots d_{t-1} c_t$ of alternating horizontal

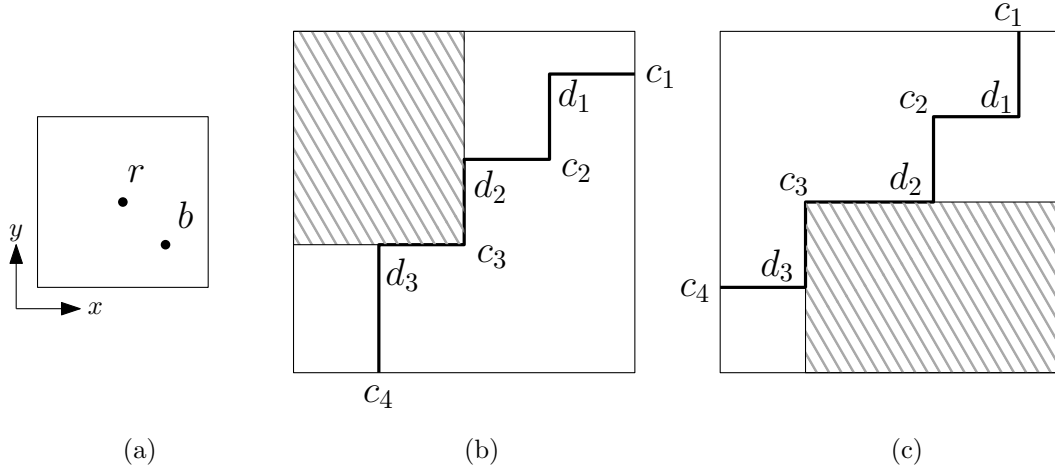


Figure 1:

line segments $c_i d_i = [x(c_i), x(d_i)] \times [y(c_i)]$ and vertical line segments $d_i c_{i+1} = [x(d_i)] \times [y(c_{i+1}), y(d_i)]$. See Figure 1(b). The points c_1, c_2, \dots, c_t are called *outward corners* and the points d_1, d_2, \dots, d_{t-1} are called *inward corners*. With each outward corner $c_i = (x, y)$, we associate a *cell* $C_i = (-\infty, x) \times (y, +\infty)$. If a point q is dominated by at least one outward corner, then q is said to lie *above* the curve \mathcal{C} . On the other hand, if a point q dominates at least one inward corner, then q is said to lie *below* the curve \mathcal{C} . The curve \mathcal{C} has the following properties:

1. Every point on curve \mathcal{C} dominates at least k points in R , but it dominates no more than $2k$ points in R .
2. If a point q dominates less than k points of R , then q lies above the curve \mathcal{C} .
3. $t = O(n/k)$, i.e., the number of cells are no more than $O(n/k)$.

LEMMA 2. *The k -shallow cutting on R can be constructed using $O(N/B)$ I/Os. The inward and the outward corners are reported in increasing order of their x -coordinate value.*

PROOF. There exists a simple algorithm to construct the k -shallow cutting on R . The details of this construction can be found in [19]. \square

Algorithm. Given R and B , we construct the first set of cells, which we call *red cells*.

1. Construct a $\lceil \frac{2K}{N} \rceil$ -shallow cutting \mathcal{C}_r on R .
2. For each blue point $b \in B$ check if it lies on/below or above the curve \mathcal{C}_r . If b lies on/below \mathcal{C}_r , then it is classified as *deep*. Otherwise, it is classified as *shallow* and b is assigned to any arbitrary cell in the cutting containing it.

3. Delete all the shallow blue points from the dataset. If the number of deep blue points are greater than $N/2$, then we report a *failure* and stop the algorithm concluding that $K^* > K$. Otherwise, we go to the next step.
4. For each cell C_i we define R_i to be the set of red points which lie in that cell, and B_i to be the set of blue points assigned to that cell.

Analysis. Now we analyze the running time of the above algorithm. Using Lemma 2, step 1 can be performed in $O(N/B)$ I/Os. Step 2 is performed as follows: for each blue point (say b) find the outward corner (say c_i) immediately to its right. b is assigned to cell C_i if it lies in the cell of C_i ; otherwise b lies on/below \mathcal{C}_r and is classified as deep. The blue points can be assigned using $O(N/B)$ I/Os since the blue points and the outward corners are already sorted along x -axis.

Next we show that when $K^* \leq K$ then the algorithm does not report a failure. Each deep blue point dominates $\geq \lceil \frac{2K}{N} \rceil$ red points (by Property 2 of shallow cuttings). Since there are at most K^* domination pairs, the number of deep blue points is $\leq \frac{K^*}{\lceil \frac{2K}{N} \rceil} \leq \frac{K}{\lceil \frac{2K}{N} \rceil} \leq N/2$. Hence, the algorithm will not report failure when $K^* \leq K$.

Now we prove that none of the three properties of the red-blue cells have been violated. For each cell C_i , the outward corner c_i dominates $O(K/N)$ red points. Therefore, $|R_i| = O(K/N)$ and hence, property (A) is not violated. By step 4 of our algorithm, we ensure property (B) for every domination pair (r, b) where b is a shallow point. The other domination pairs will be taken care of in the next steps.

Since each point in B is assigned to exactly one cell, $\sum_{i=1}^{\ell} |B_i| \leq N$. By Property 3 of shallow cuttings, the number of cells constructed is $O(N^2/K)$, and by Property 1 of shallow cuttings, each cell contains $O(K/N)$ red points. Therefore, $\sum_{i=1}^{\ell} |R_i| = O(N^2/K) \times O(K/N) = O(N)$. Therefore, property (C) has not yet been vio-

lated.

Remark. Note that a red point in R can belong to many R_i 's, whereas a shallow blue point in B will belong to exactly one B_i .

3.2 Second step: Blue cells

After the first step, all the domination pairs (r, b) involving the shallow blue points have been taken care of. In the next two steps, we discuss how to build additional cells which will capture domination pairs involving the deep blue points.

We will use shallow cuttings again, but this time we will change the orientation of our cells. A k -shallow cutting on the deep blue points is a curve $\mathcal{C} = c_1 d_1 c_2 d_2 \dots d_{t-1} c_t$ of alternating vertical line segments $c_i d_i = [x(c_i)] \times [y(d_i), y(c_i)]$ and horizontal line segments $d_i c_{i+1} = [x(c_{i+1}), x(d_i)] \times [y(d_i)]$. See Figure 1(c). The points c_1, c_2, \dots, c_t are called *outward corners* and the points d_1, d_2, \dots, d_{t-1} are called *inward corners*. With each outward corner $c_i = (x, y)$, we associate a *blue cell* $C_i = (x, \infty) \times (-\infty, y)$. If a point q is dominated by at least one inward corner, then q is said to lie *above* the curve \mathcal{C} . On the other hand, if a point q dominates at least one outward corner, then q is said to lie *below* the curve \mathcal{C} . The curve \mathcal{C} should have the following properties:

1. Every point on curve \mathcal{C} is dominated by at least k deep blue points, but it is dominated by no more than $2k$ deep blue points.
2. If a point q is dominated by less than k deep blue points, then q lies below the curve \mathcal{C} .
3. $t = O(n/k)$.

Algorithm. Let B_D be the set of deep blue points. Given R and B_D , the following steps are performed:

1. Construct $\lceil \frac{2K}{N} \rceil$ -shallow cutting \mathcal{C}_b on all the deep blue points.
2. For each red point r check if it lies on/above or below the curve \mathcal{C}_b . If r lies on/above \mathcal{C}_b , then it is classified as *deep*. Otherwise, it is classified as *shallow* and r is assigned to any arbitrary cell in the cutting containing it.
3. If the number of deep red points are greater than $N/2$, then we report a *failure* and stop the algorithm concluding that $K^* > K$. Otherwise, we go to the next step.
4. For each cell C_i we define B_i to be the set of deep blue points which lie in that cell, and R_i to be the set of red points assigned to that cell.

Following the analysis from the previous step, the number of I/Os performed in this step is also bounded by $O(N/B)$, and it can be shown that none of the properties of the red-blue cells are violated yet.

3.3 Third step: Recursion

In the second step, all the domination pairs (r, b) such that r is a shallow red point and b is a deep blue point will be taken care of. After the first two steps, now we are left with deep red points R_D and deep blue points B_D . We know that $|R_D| < N/2$ and $|B_D| < N/2$; else a failure would have been reported.

Algorithm. Recurse on R_D and B_D , and all occurrences of N in the algorithm are replaced with $N/2$. The algorithm stops when the red and the blue set is smaller than a suitable constant C .

Let $T(N)$ denote the total number of I/Os performed by this algorithm. Then,

$$T(N) \leq \begin{cases} O(1) & \text{if } n \leq C; \\ O\left(\frac{N}{B}\right) + T(N/2) & \text{otherwise.} \end{cases}$$

Solving this recurrence we get $T(N) = O\left(\frac{N}{B}\right)$. By a similar recurrence, Property (C) of red-blue cells is satisfied. It is easy to verify that Property (A) and (B) are also satisfied. This finishes the proof of Lemma 1.

4. THE ADAPTIVE ALGORITHM

Now we are ready to prove Theorem 1.

4.1 First step: A non-adaptive algorithm

The first step in building our adaptive solution is the construction of a *non-adaptive* algorithm.

THEOREM 3. *Consider a list R of N_r elements and a list B of N_b blue elements. Then there exists a non-adaptive algorithm for red-blue dominance counting problem which requires $O\left(\frac{N}{B} \log_{M/B}\left(\frac{\min\{N_r, N_b\}}{B}\right)\right)$ I/Os, where $N = N_r + N_b$.*

PROOF. We will only give a high-level description of this algorithm. Most of the details are fairly standard. Without loss of generality, assume that $N_r = \min\{N_r, N_b\}$. As in distribution sort, in $O(N_r/B)$ I/Os the list R is split into $\Theta\left(\sqrt{\frac{M}{B}}\right)$ lists R_1, R_2, \dots, R_f of roughly equal size, such that for any $i < j$, any element in R_i is smaller than any element in R_j . The order of the elements in any R_i is systematic with their order in R . An element b is defined to *belong* to a set R_i if the value of the blue element lies between the value of the smallest and the largest element in R_i . By performing a synchronized scan of all the R_i 's, in $O(N_b/B)$ I/Os, for each element in B (say it belongs to R_i) we can compute the number of red points in $\bigcup_{i=1}^f R_i$ it dominates. Finally, $\forall i \in [1, f]$, we recurse on R_i and the set of blue points which belong to R_i . The number of levels of recursion will be $O(\log_{M/B} \frac{N_r}{B})$. \square

4.2 Second step: K -capped structure

Now we will solve the *K -capped red-blue dominance counting* problem: Given a set R of N red points, a set B of N blue points, and a value K , we need to compute

K^* , but if $K^* > K$, then we are allowed to report *failure*. We will prove the following result.

THEOREM 4. *K -capped red-blue dominance counting problem can be solved using $O\left(\frac{N}{B} + \frac{N}{B} \log_{M/B}\left(\frac{K}{NB}\right)\right)$ I/Os.*

Now we prove Theorem 4.

Algorithm. Using Lemma 1, construct red-blue cells on R and B with parameter K . If Lemma 1 reports a failure, then we stop the algorithm. Otherwise, we obtain a set of cells C_1, \dots, C_ℓ . For each $i \in [1, \ell]$, based on R_i and B_i associated with C_i , we run the non-adaptive algorithm of Theorem 3. Finally, add up the count obtained from all the cells.

Analysis. The number of I/Os performed will be bounded by

$$\begin{aligned} & \sum_{i=1}^{\ell} O\left(\left(\frac{|R_i| + |B_i|}{B}\right) \log_{M/B}\left(\frac{\min\{N_r, N_b\}}{B}\right)\right) \\ & \leq \left(\log_{M/B} \frac{K}{NB}\right) \sum_{i=1}^{\ell} O\left(\frac{|R_i| + |B_i|}{B}\right) \quad \text{by property (A)} \\ & \leq O\left(\frac{N}{B} \log_{M/B}\left(\frac{K}{NB}\right)\right) \quad \text{by property (C)} \end{aligned}$$

4.3 Third step

Using a trick from the computational geometry literature, the solution to the K -capped red-blue dominance counting problem (Theorem 4) can be used to efficiently solve the red-blue dominance counting problem (Theorem 1).

We use Chan's guessing trick from [5]. The algorithm is executed as a series of rounds. In round i (starting from $i = 1$), we construct the K_i -capped structure of Theorem 4 for

$$K_i = (NB) \cdot \left(\frac{M}{B}\right)^{2^i}$$

If the algorithm returns the value of K^* , then we are done and the algorithm terminates. Otherwise, we proceed to round $i + 1$. Let j be the number of rounds performed before termination. If $j = 1$ then the number of I/Os performed is $O(N/B)$. Otherwise, if $j > 1$ then in round $j - 1$ since we reported failure, $K^* > (NB) \cdot \left(\frac{M}{B}\right)^{2^{j-1}} \implies 2^j < 2 \log_{M/B} \frac{K^*}{NB}$. The total number of I/Os performed in all the j rounds is bounded by $\sum_{i=1}^j O\left(\frac{N}{B} \log_{M/B}\left(\frac{K_i}{NB}\right)\right) = \sum_{i=1}^j O\left(\frac{N}{B} \cdot 2^i\right) = O\left(\frac{N}{B} \cdot 2^j\right) = O\left(\frac{N}{B} \log_{M/B}\left(\frac{K^*}{NB}\right)\right)$.

Remark. This algorithm can be modified to match the state of the art adaptive algorithms for the comparison based model [17] and the RAM model [10]. This involves replacing the non-adaptive I/O-model algorithm of Theorem 3 with the non-adaptive algorithm in the

comparison based model which takes $O(N \log N)$ time and the non-adaptive algorithm in the RAM model [6].

5. THE APPROXIMATION ALGORITHM

In this section we will prove Theorem 2. Our solution is based on an interesting combination of random sampling and red-blue cells. The number of domination pairs, K^* , can lie in the range $[0, N^2]$. We will split the solution into three different cases and handle each of them separately.

5.1 When $K^* \in [0, N]$

By setting M and B to be appropriate constants, the I/O-model solution of Theorem 4 maps to the RAM model. We obtain the following result.

LEMMA 3. *K -capped red-blue dominance counting problem can be solved in $O(N + N \log_2(\frac{K}{N}))$ time in the RAM model.*

Using Lemma 3 with $K = N$, we can either obtain the exact number of inversions in $O(N)$ time, or it will report a failure which implies that $K^* > N$.

5.2 When $K^* \in [N, N\sqrt{N} \log N]$

Algorithm. The following steps are performed:

(1) Construct the red-blue cells for parameter $K = N\sqrt{N} \log N$ using Lemma 1. If a failure is reported, then we conclude that $K^* > N\sqrt{N} \log N$ and stop the algorithm. Otherwise, go to the next step.

(2) Pick N samples. Each sample is a pair (r, b) such that if $b \in B_i$ then $r \in R_i$. Each sample is picked by the following three stage process:

1. Pick a set B_i . A set B_i is sampled with probability $\frac{|R_i||B_i|}{\sum_{i=1}^{\ell} |R_i||B_i|}$.
2. Sample a point in B_i . Each point in B_i is sampled with probability $\frac{1}{|B_i|}$.
3. Sample a point in R_i . Each point in R_i is sampled with probability $\frac{1}{|R_i|}$.

(3) Let X be the number of samples which are domination pairs. Then we report $X \cdot C\sqrt{N} \log N$ as the answer, where the constant C is defined later.

LEMMA 4. *Consider a pair (r, b) such that $r \in R_i$ and $b \in B_i$. The probability of the pair (r, b) being picked is $\frac{1}{\sum_{i=1}^{\ell} |R_i||B_i|}$, i.e., each pair is picked with equal probability.*

PROOF. The probability of the pair (r, b) being picked is $\frac{|R_i||B_i|}{\sum_{i=1}^{\ell} |R_i||B_i|} \times \frac{1}{|B_i|} \times \frac{1}{|R_i|} = \frac{1}{\sum_{i=1}^{\ell} |R_i||B_i|}$ \square

LEMMA 5. *The sample space is $O(N\sqrt{N} \log N)$. In other words, $\sum_{i=1}^{\ell} |R_i||B_i| = O(N\sqrt{N} \log N)$.*

PROOF. We split the summation $\sum_{i=1}^{\ell} |R_i||B_i|$ into two disjoint summations: one in which $|R_i| = \min\{|R_i|, |B_i|\}$,

and other one in which $|B_i| = \min\{|R_i|, |B_i|\}$. Consider the first summation:

$$\begin{aligned} \sum_{i=1}^{\ell} |R_i| |B_i| &\leq O(K/N) \sum_{i=1}^{\ell} |B_i| \quad \text{by property (A)} \\ &\leq O(K/N) \cdot O(N) \quad \text{by property (C)} \\ &= O(N\sqrt{N} \log N) \end{aligned}$$

The same bound can be shown for the other summation as well. \square

LEMMA 6. *For a fixed constant c , with high probability $1 - 1/N^c$, the estimate will lie in the range*

$$\left[\left(1 - \frac{\log N}{N^{1/4}}\right) K^*, \left(1 + \frac{\log N}{N^{1/4}}\right) K^* \right].$$

PROOF. Recall that X is the number of domination pairs picked in the N samples. For $i \in [1, N]$, define $X_i = 1$ if the i -th sample picked is a domination pair; otherwise $X_i = 0$. Therefore, $X = \sum_{i=1}^N X_i$. The expected value of X , i.e., $E[X]$ will be equal to $N \cdot \frac{K^*}{\sum_{i=1}^{\ell} |R_i| |B_i|} = N \cdot \frac{K^*}{C N \sqrt{N} \log N} = \frac{K^*}{C \sqrt{N} \log N}$, where C is the constant inside $O(N\sqrt{N} \log N)$.

To apply Chernoff bounds, we need to perform the following set of calculations. Set a parameter $\varepsilon = \frac{\log N}{N^{1/4}}$ and use the fact that $K^* \geq N$, to observe that

$$\varepsilon^2 E[X] = \varepsilon^2 \frac{K^*}{C \sqrt{N} \log N} > \varepsilon^2 \frac{\sqrt{N}}{C \log N} = \frac{\log N}{C}$$

By applying Chernoff bounds, we get

$$\Pr \left[\left| X - E[X] \right| > \varepsilon E[X] \right] < e^{-\Omega(\varepsilon^2 E[X])} < e^{-\Omega(\log N)} < N^{-c}$$

\square

5.3 When $K^* \in [N\sqrt{N} \log N, N^2]$

Algorithm. The following steps are performed:

- (1) Pick N random samples. Each sample is of the form (r, b) where $r \in R$ and $b \in B$. Each red point in R is picked with probability $\frac{1}{N}$ and each blue point in B is picked with probability $\frac{1}{N}$.
- (2) Let X be the number of samples which are domination pairs. Then we report $X \cdot N$ as the answer.

LEMMA 7. *Let c be a sufficiently large constant. Then with high probability $1 - 1/N^c$, the estimate will lie in the range $\left[\left(1 - \frac{1}{N^{1/4}}\right) K^*, \left(1 + \frac{1}{N^{1/4}}\right) K^* \right]$.*

PROOF. Let X be the number of domination pairs picked in the N samples. For $i \in [1, N]$, define $X_i = 1$ if the i -th sample picked is a domination pair; otherwise $X_i = 0$. Therefore, $X = \sum_{i=1}^N X_i$. Now, $E[X] = N \cdot \frac{K^*}{N^2} = \frac{K^*}{N}$.

To apply Chernoff bounds, we need to perform the following set of calculations. Set a parameter $\varepsilon = 1/N^{1/4}$

and use the fact that $K^* \geq N\sqrt{N} \log N$, to observe that

$$\varepsilon^2 E[X] = \varepsilon^2 \frac{K^*}{N} > \varepsilon^2 \frac{N\sqrt{N} \log N}{N} = \log N$$

By applying Chernoff bounds, we get

$$\Pr \left[\left| X - E[X] \right| > \varepsilon E[X] \right] < e^{-\Omega(\varepsilon^2 E[X])} < e^{-\Omega(\log N)} < N^{-c}$$

\square

6. REFERENCES

- [1] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 180–186, 2009.
- [2] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988.
- [3] Miklós Ajtai, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 370–379, 2002.
- [4] Arne Andersson and Ola Petersson. Approximate indexed lists. *Journal of Algorithms*, 29(2):256–276, 1998.
- [5] Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.
- [6] Timothy M. Chan and Mihai Patrascu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010.
- [7] Persi Diaconis and R. L. Graham. Spearman’s footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(2):262–268, 1977.
- [8] Paul F. Dietz. Optimal algorithms for list indexing and subset rank. In *Algorithms and Data Structures Workshop (WADS)*, pages 39–46, 1989.
- [9] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of International World Wide Web Conferences (WWW)*, pages 613–622, 2001.
- [10] Amr Elmasry. Counting inversions adaptively. *CoRR*, abs/1503.01192, 2015.
- [11] Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.
- [12] Anupam Gupta and Francis Zane. Counting inversions in lists. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 253–254, 2003.

- [13] Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004.
- [14] Yijie Han and Mikkel Thorup. Integer sorting in $O(n \sqrt{\log \log n})$ expected time and linear space. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 135–144, 2002.
- [15] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.
- [16] Jirí Matousek. Reporting points in halfspaces. *Computational Geometry*, 2:169–186, 1992.
- [17] Kurt Mehlhorn. Sorting presorted files. In *Theoretical Computer Science, 4th GI-Conference, Aachen, Germany, March 26-28, 1979, Proceedings*, pages 199–212, 1979.
- [18] Anna Pagh, Rasmus Pagh, and Mikkel Thorup. On adaptive integer sorting. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 556–579, 2004.
- [19] Darren Erik Vengroff and Jeffrey Scott Vitter. Efficient 3-d range searching in external memory. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 192–201, 1996.

Appendix: I/O-model

In this model [2], a machine is equipped with M words of main memory, and a disk that has been formatted into *blocks* of B words each. The values of M and B satisfy $M \geq 2B$. An I/O either reads a disk block into memory, or writes B words of memory into a disk block. The *time* of an algorithm is measured in the number of I/Os performed, while the *space* is measured in the number of disk blocks occupied.